README for the Python code and downstream R processing to the paper *"The epigenome and regulatory proteins are highly redundant and linearly related"*

Tobias Ahsendorf[1,3], Ved Topkar[2,3], Jeremy Gunawardena[3] & Roland Eils[1]

[1]DKFZ, D-69120 Heidelberg, Germany
[2]Harvard College, Cambridge, MA, USA
[3]Department of Systems Biology, Harvard Medical School, Boston, MA, USA

April 9, 2016

This README is split into two parts; first we give an overview over all Python scripts and later on we show how these scripts might be used to generate the appropriate csv files and how these files can be processed and pasted together in R.
Our scripts require the use of two non-standard Python packages, HTSeq and Wiggelen, which can be downloaded at https://pypi.python.org/pypi/HTSeq/ and https://pypi.python.org/pypi/wiggelen/, respectively.

1. `GTF_processing.py`
   processes a GTF file (e.g., from GENCODE), extracts all relevant knowledge in terms, of Gene_ID, Transcript_ID, Gene_type, Start/End position of transcript and strand information and stores the result in a csv file.
   Input arguments:

   (a) Path to the GTF file

   (b) Path where the CSV file should be stored.

   Example:

   ```
   python GTF_processing.py /home/user/gencode.v18.
       annotation.gtf.gz        /home/user/Transcripts-
        v.18.csv
   ```

   Output:
   A CSV file where to each transcript in the GTF file the Gene_ID, Transcript_ID, Gene_type, Start/End position of transcript and strand information are listed.

2. `counting_the_number_of_cpgs.py` (optional, when whole genome bisulfite-sequencing DNA methylation data are considered)
   processes a human genome Fasta file (e.g., by downloading the hg19.2bit file from here and converting it to a Fasta file with `twoBitToFa` (see here for downloading `twoBitToFa` and converting it with something like `twoBitToFa hg19.2bit hg19.fa`)) and counts the number of CpGs per base pair around TSSs, in Transcripts and around TTSs for various bin resolutions for protein coding or lincRNA genes etc. and stores the results as csv files.
   Input arguments:

   (a) Path to the GTF CSV file (the output from `GTF_processing.py`)
   (b) Path to human genome Fasta file
   (c) type ("protein˙coding" or "lincRNA" etc.)
   (d) halfwinwidth is the range, e.g., +/- halfwinwidth around the TSS/TTS..., suggested value 2000
   (e) granulationwin is the vector for numbers of windows the area around the TSS/TTS or the entire transcript is split into, suggested 1 and 40, Input: "1,40"
   (f) Folder to store profile matrix, as input you may take, e.g., "/home-/user/" - here one may get a feel of the average values per position (TSS etc.) and per bin, which gives for higher bin resolutions a good feeling for the average profile
   (g) Folder to store CpG count matrices, as input you may take, e.g., "/home/user/"

   Example:

   ```
   python counting_the_number_of_cpgs.py  /home/user/
      Transcripts-v.18.csv  /home/user/hg19.fa
      protein_coding 2000 1,40 /home/user/ /home/user
      /
   ```

   Output:
   Profile CSV file (see input argument no. 6), named, e.g., "lincRNA.csv" in folder of input argument no. 6. Count CSV files (see input argument no. 7), named e.g., "lincRNA_TSS_1.csv in folder of input argument no. 7.

3. `BEDGRAPH-processing.py` and `WIG-processing.py`
   take BedGraph or WIG files as input and count the number of tags per base pair around TSSs, in Transcripts and around TTSs for various bin resolutions for protein coding or lincRNA genes etc. and stores the results as csv files. Since many ENCODE files are BigWig files one may convert them to BedGraph files with `bigWigToBedGraph` (see here for downloading this tool).
   Input arguments:

   (a) Path to the GTF CSV file (the output from `GTF_processing.py`)
   (b) Path to BedGraph or Wig file

(c) type ("protein˙coding" or "lincRNA" etc.)

(d) halfwinwidth is the range, e.g., +/- halfwinwidth around the TSS/TTS..., suggested value 2000

(e) granulationwin is the vector for numbers of windows the area around the TSS/TTS or the entire transcript is split into, suggested 1 and 40, Input: "1,40"

(f) Folder to store profile matrix, as input you may take, e.g., "/home-/user/"

(g) Folder to store tag count matrices, as input you may take, e.g., "/home/user/"

(h) Name for the different files to be stored

(i) optional for `BEDGRAPH-processing.py`: txt file where we can see whether or not the file format has been ok for our purposes (works at least for BigWig ENCODE file that have been transformed to BedGraph via the `bigWigToBedGraph` UCSC tool)

Example:

```
python BEDGRAPH-processing.py  /home/user/
   Transcripts-v.18.csv  /home/user/example.bg
   protein_coding 2000 1,40 /home/user/ /home/user
   / test_name /home/user/corrext_type.txt
```

Output:
Profile CSV file (see input argument no. 6), named, e.g., "test_name.csv" in folder of input argument no. 6. Tag count CSV files (see input argument no. 7), named e.g., "test_name_TSS_1.csv" in folder of input argument no. 7.
Note:
In order that the R scripts can be used for data processing it is beneficial if for one cell line and one transcript type all tag count CSV files are contained in one folder.

4. `remove_genes_from_blacklist_and_chromosomes.py` (optional, if genes should/must not be considered)
outputs which genes for protein coding resp. lincRNA genes etc. overlap with regions which should not be considered, for which we may, for instance, take the ENCODE consensus blacklist (see here for further information, and to download the blacklist file see here) and chromosomes like chrX, chrY and chrM.
Input arguments:

(a) Path to the GTF CSV file (the output from `GTF_processing.py`)

(b) Path to the consensusBlacklist or any other blacklist (BED format)

(c) type ("protein_coding" or "lincRNA" etc.)

(d) Folder to store txt file with genes not to be considered, as input you may take, e.g., "/home/user/"

(e) optional 5th to n-th input argument(s): chromosomes, for which all genes, that lie on these chromosomes, should not be considered, e.g., "chrX chrY chrM"

Example:

```
python remove_genes_from_blacklist_and_chromosomes
    .py /home/user/Transcripts-v.18.csv   /home/
    user/consensusBlacklist.bed protein_coding /
    home/user/ chrX chrY chrM
```

Output:
Text file, named, e.g., "protein_coding_Not_to_be_considered.txt" in folder of input argument no. 4, where in each line we do have a gene that has an overlap with the blacklist regions.

A workflow might look as follows:
We first download a GENCODE GTF file or any other gene annotation file and call `GTF_processing.py` as outline above.
When whole genome bisulfite-sequencing DNA methylation data are considered in our analysis (and it is beneficial to normalize them by the CpG content of the respective loci), then we call `counting_the_number_of_cpgs.py` as outlined above.
Now for the BigWig, BedGraph and Wig files, we will process them with `BEDGRAPH-processing.py` and `WIG-processing.py` as outlined above.
Once all these files are generated we can proceed in R:

```
#load all functions from functions.R
source('/home/functions.R')
#for details of the functions see the comments made in functions.R

#if you want to paste all data together for a particular gene type,
#loci (TSS/Transcript/TTS) and cell line:
#paths to the individual csv files for the epigenetic marks
#for a particular gene type and cell line (the output of
#BEDGRAPH-processing.py and WIG-processing.py), e.g.,
path = c("/home/test_TSS_1.csv","/home/test2_TSS_1.csv")
#or path = c("/home/test_TSS_40.csv","/home/test2_TSS_40.csv")
#or path = c("/home/test_TTS_1.csv","/home/test2_TTS_1.csv")

#marks at the respective positions of paths, e.g.,
marks =  c("H3K4me3","DNA meth")

#to obtain the pasted data frame we call
X = paste_processed_data(paths,marks)
#or with a different quantile cutoff
X = paste_processed_data(paths,marks,quantile_cutoff=0.05)
#if you want to add a blacklist, say
blacklist_path = "/home/protein_coding_Not_to_be_considered.txt"
#you can call, in order to remove the desired genes,
X = paste_processed_data(paths,marks,blacklist_path=blacklist_path)
```

```
#if you want to normalize the DNA methylation by the no. of CpGs
#you first have to specify a path to it, e.g.,
cpg_path = "/home/protein_coding_TSS_1.csv"
#or cpg_path = "/home/protein_coding_TSS_40.csv"
#or cpg_path = "/home/protein_coding_TTS_1.csv"
#then you have to specify a vector of entries which you want
#to normalize, e.g,
dna_normalization_vec = 2
#since in our case the DNA methylation is assumed to in the
#second position of paths and marks
#also you can set dna_normalization_vec = 1:2 or similarly,
#in case there are more positions you want to normalize

#finally you call
X = paste_processed_data(paths,mark_names,blacklist_path=blacklist_path,
dna_normalization_vec=dna_normalization_vec,cpg_path=cpg_path)

#if you want to calculate the Cage gene expression as we did,
#see functions.R for details:

#paths to the plus and minus strand Cage csv files
#for a particular gene type and cell line (the output of
#BEDGRAPH-processing.py and WIG-processing.py), e.g.,
path = c("/home/plus_TSS_1.csv","/home/minus_TSS_1.csv")
#or path = c("/home/plus_TSS_40.csv","/home/minus_TSS_40.csv")
#in any event it should be centric around the TSSs
#and call
cage = combine_cage_data(paths)

#if you want to modify the pseudo count call something like
cage = combine_cage_data(paths,pseudo_count=0.1)

#if you want to include the blacklist (we take the one from above)
#call
cage = combine_cage_data(paths,blacklist_path=blacklist_path)
```